**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники

и автоматизированных систем

**КУРСОВАЯ РАБОТА**

по дисциплине «Основы программирования»

Тема: Метро и поиск оптимального пути

Автор работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Донцов А.А.

(подпись)

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Брусенцева В.С.

(подпись)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Белгород

2018 г.

Оглавление

[Введение 3](#_Toc533252868)

[Постановка задачи 3](#_Toc533252869)

[Метод решения задачи 4](#_Toc533252870)

[Структуры данных 6](#_Toc533252871)

[Алгоритм решения задачи: 7](#_Toc533252872)

[Описание подпрограмм 7](#_Toc533252873)

[Тестовые данные 22](#_Toc533252874)

[Модули 23](#_Toc533252875)

[Заключение 24](#_Toc533252876)

[Используемая литература 24](#_Toc533252877)

[Приложение 24](#_Toc533252878)

# Введение

В жизни люди всегда сталкиваются с проблемой поиска кратчайшего пути. Так, например, в больших городах люди пользуются метрополитеном, в котором очень большое количество станций. Из-за этого, очень часто требуется выбирать самый оптимальный путь движения, самый быстрый и самый удобный. Поэтому эта проблема достаточна интересна и актуальна.

# Постановка задачи

Разработка алгоритма и создание модульной программы для решения следующей задачи:

Пусть есть схема метро. Расстояние между станциями известно.

Надо найти оптимальный путь между двумя станциями, где под оптимальным путём понимать один из вариантов:

1. Наименьшее количество пересадок

2. Наименьшее число станций по пути следования (при равенстве числа станций минимизировать число пересадок)

3. Наименьшее время в пути (скорость движения между каждой парой станций известна как в ту, так и в другую сторону; время на пересадку известно тоже)



# Метод решения задачи

Задача решается с использованием гибкой и универсальной структуры данных под названием граф, которую очень часто используют программисты для определения оптимальных способов решения проблемы, в нашем случае оптимальный путь от станции A до станции B.

Дадим определение графа.

Пусть V – непустое множество, – множество всех его двухэлементных подмножеств. Пара (V,E), где называется графом(неориентированным графом)

Граф называется конечным, если множество V конечно.

Элементы множества V называются вершинами графа, а элементы множества E называются ребрами графа.

Число вершин графа G = (V,E) называется его порядком.

Исходя из определения графа, каждому ребру соответствует двухэлементное подмножество вершин. Если подмножество {} определяет ребро e, то ребро e инцидентно вершинам а вершины смежны. Вершины называются концевыми вершинами ребра e. Два ребра, инцидентные одной вершине, называются смежными. Таким образом, смежность есть отношение между однородными элементами графа, а инцидентность – отношение между разнородными элементами графа.

Множество вершин, смежных с вершиной , называется множеством смежности вершины и обозначается . Если , то Г(A) – множество всех вершин, смежных с вершинами из A.

Если ребра графа имеют какие-либо показатели (в нашем случае время прохода между вершинами), то он называется взвешенным.

Представим, схему метро, как ориентированный взвешенный граф, тогда мы получим связанную структуру из станций, весами дуг будут значения для поиска оптимального пути между станциями.

Если станции соединены железнодорожными путями, назовём это связью.



Под пересадкой будем понимать переход между графами, используя информацию об общих станциях. Пример общих станций Площадь Александра Невского – 1 и Площадь Александра Невского – 2.



Для поиска оптимального пути будем использовать Алгоритм Дейкстры. Алгоритм, в зависимости от оптимального пути будет запоминать данные перехода от начальной станции до остальных станций метро, а именно, результатом выполнения программы будут оптимальные пути до всех станций. Принцип заключается в том, что Алгоритм сохраняет все веса от начального графа до соседей. Находит ближайший граф и повторяет поиск соседей, если будет пересечение путей, то сохраняется кратчайший путь, а также сохраняется предыдущая станция из которой получился данный путь.

Каждой станции присвоим свой номер, свою вершину графа, обращаться к станции будем по этому номеру.

1.Автово

2.Адмиралтейская

3.Балтийская

4.Бухарестская

5.Василеостровская

6.Владимирская

7.Волковская

8.Выборгская

9.Горьковская

10.Гостиный двор

11.Гражданский проспект

12.Девяткино

13.Достоевская

14.Елизаровская

15.Звёздная

16.Звенигородская

17.Кировский завод

18.Комендантский проспект

19.Крестовский остров

20.Купчино

21.Ладожская

22.Ленинский проспект

23.Лесная

24.Лиговский проспект

25.Ломоносовская

26.Маяковская

27.Международная

28.Московская

29.Московские ворота

30.Нарвская

31.Невский проспект

32.Новочеркасская

33.Обводный канал

34.Обухово

35.Озерки

36.Парк Победы

37.Парнас

38.Петроградская

39.Пионерская

40.Площадь Александра Невского 1

41.Площадь Александра Невского 2

42.Площадь Восстания

43.Площадь Ленина

44.Площадь Мужества

45.Политехническая

46.Приморская

47.Пролетарская

48.Проспект Большевиков

49.Проспект Ветеранов

50.Проспект Просвещения

51.Пушкинская

52.Рыбацкое

53.Садовая

54.Сенная площадь

55.Спасская

56.Спортивная

57.Старая Деревня

58.Технологический институт 1

59.Технологический институт 2

60.Удельная

61.Улица Дыбенко

62.Фрунзенская

63.Чёрная речка

64.Чернышевская

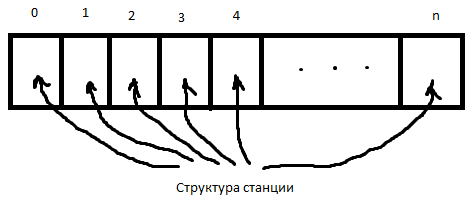
65.Чкаловская

66.Электросила

67.Академическая

# Структуры данных

Для хранения информации о станциях мы будем использовать динамический массив структур (размер будет равен количеству станций Метрополитена):



typedef struct {//структура станции

char name[MAXSTRLN];//название станции

int \*connection; // указатель на станции связи

int c\_connection;//кол-во связей со станциями

int \*together;// указатель на общие станции

int c\_together;//кол-во общих станций

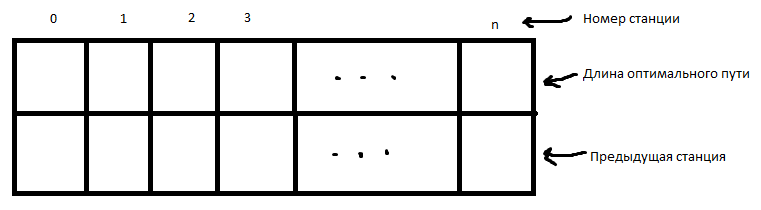
} station;

В указателе на станции связи connection и указателе на общие станции together будем выделять память, для массива размера c\_connection и c\_together соответственно. В массивах будем хранить станции, на которые можно попасть из станции данной структуры.

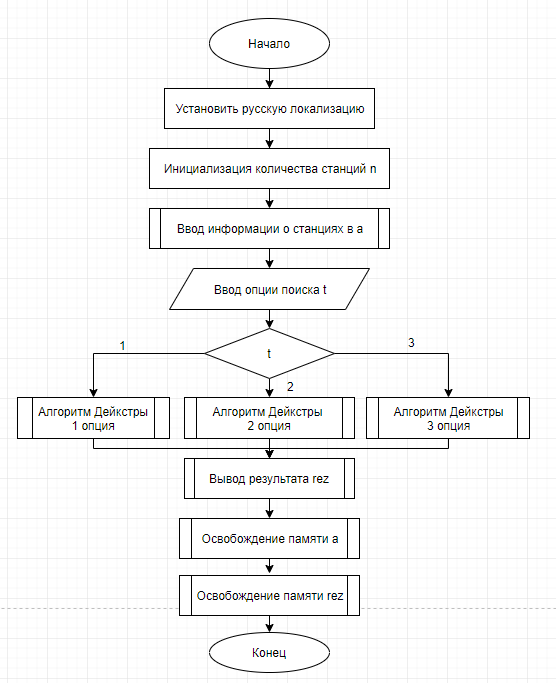
В c\_connection и с\_together храниться количество станций связей и общих станций.

Информацию о результате выполнения программы будем хранить в динамическом массиве размера 2\*n, где n - количество станций. Здесь будем хранить путь от начальной станции до конечной и информацию о длине оптимального пути. Номер станции получаем прибавлением единицы к индексу.

Длина оптимального пути до станции по индексу находится в первой половине массива (от 0 до n). Для того, чтобы получить путь от начальной станции до конечной будем хранить предыдущую станцию в пути от станции по индексу.



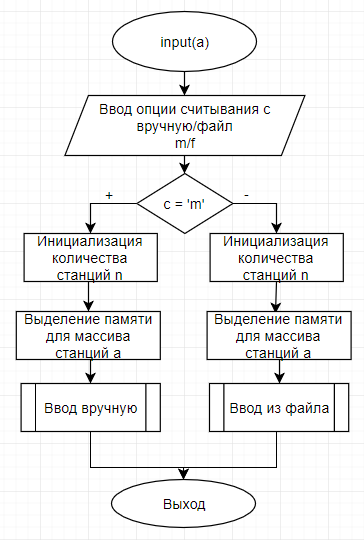
# Алгоритм решения задачи:



# Описание подпрограмм

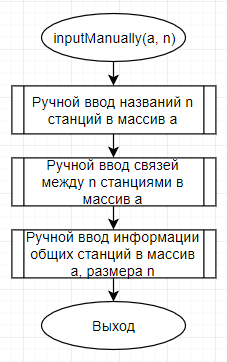
**Заголовок:** void input(station \*a, int n)

**Назначение:** Ввод информации о станциях в массив списков a, размера n



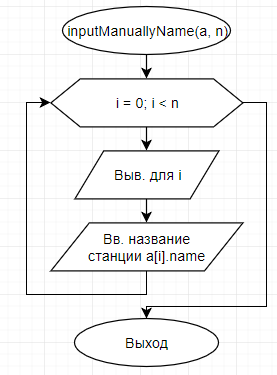
**Заголовок:** void inputManually(station \*a, int n)

**Назначение:** Ввод вручную в массив станций a, размера n



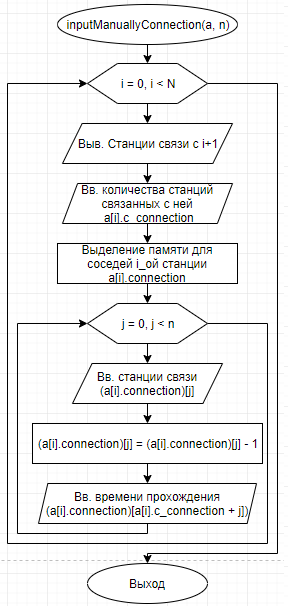
**Заголовок:** void inputManuallyName(station \*a, int n)

**Назначение:** Ввод вручную n\_ое количество названий станций в массив списков a



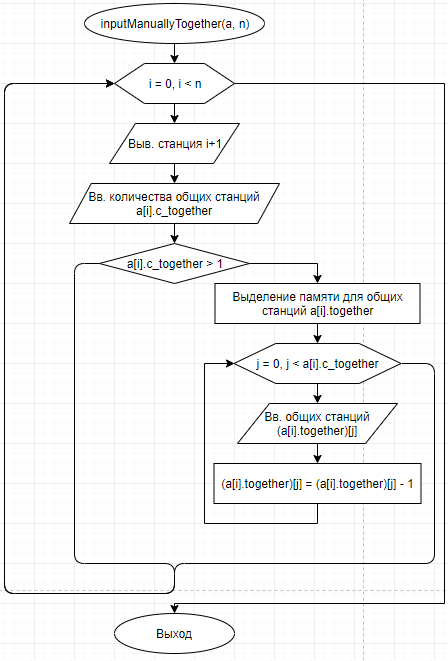
**Заголовок:** void inputManuallyConnection(station \*a, int n)

**Назначение:** Ввод связей между станциями в массив списков a, размера n из консоли



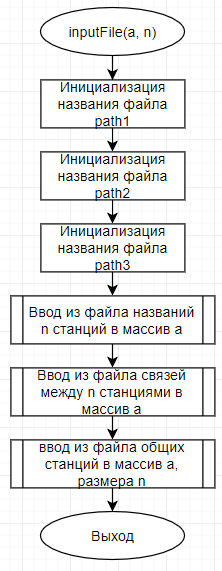
**Заголовок:** void inputManuallyTogether(station \*a, int n)

**Назначение:** Ввод общих станций в массив списков a, размера n из консоли



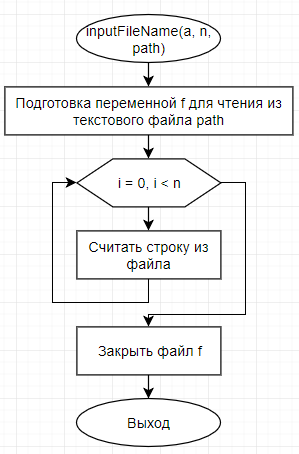
**Заголовок:** void inputFile(station \*a, int n)

**Назначение:** Ввод в массив станций a, размера n из файла



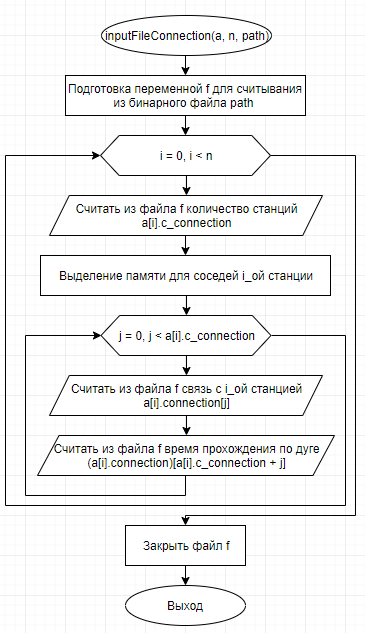
**Заголовок:** void inputFileName(station \*a, int n)

**Назначение:** Ввод названий станций в массив списков a, размера n из файла



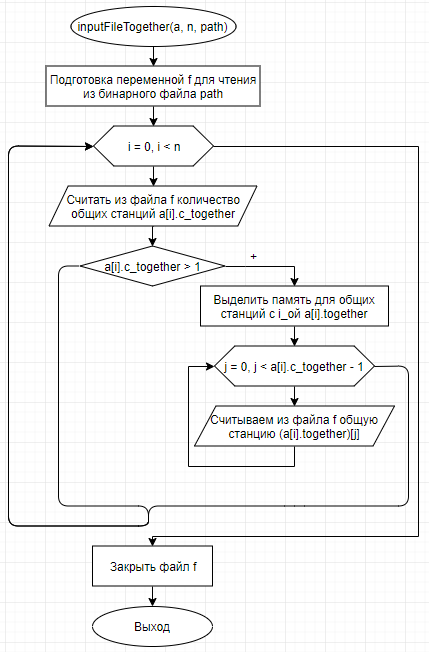
**Заголовок:** void inputFileConnection(station \*a, int n)

**Назначение:** Ввод связей между станциями в массив списков a, размера n из файла



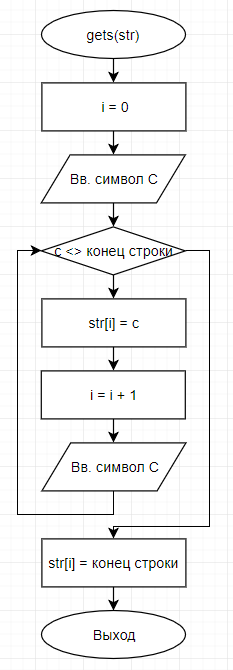
**Заголовок:** void inputFileTogether(station \*a, int n)

**Назначение:** Ввод общих станций в массив списков a, размера n из файла



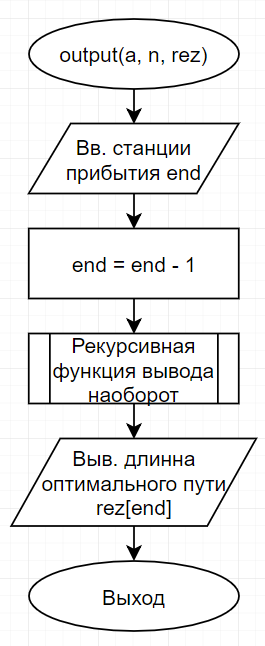
**Заголовок:** void gets(char \*str)

**Назначение:** Ввод строки с консоли в str



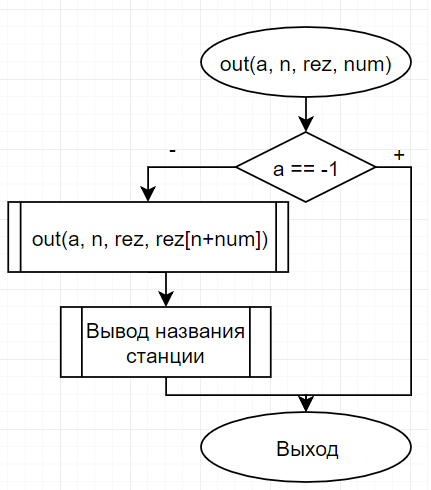
**Заголовок:** void output(station \*a, int n, int \*rez)

**Назначение:** Вывод результата rez массива станций a, размера n



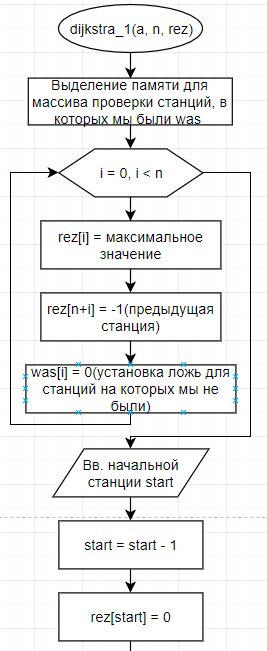
**Заголовок:** void out(station \*a, int n, int \*rez, int num)

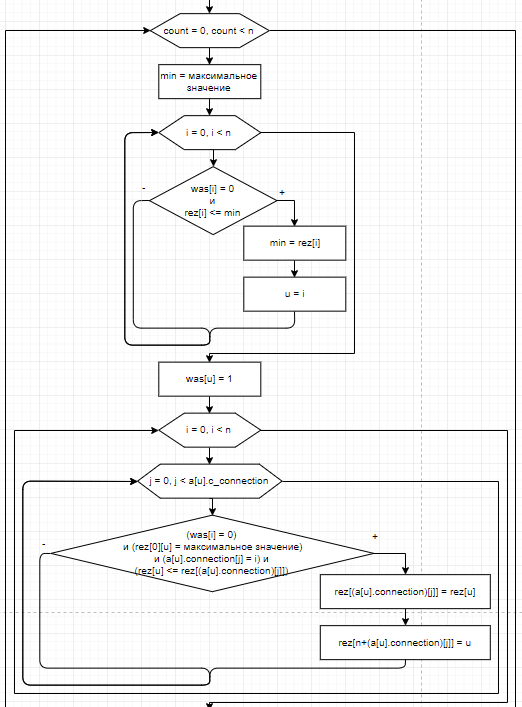
**Назначение:** Вывод оптимального пути

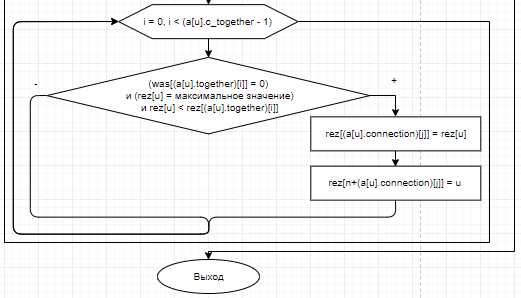


**Заголовок:** void dijkstra\_1(station \*a, int n, int \*rez)

**Назначение:** Записывает в rez путь с наименьшим количеством пересадок из массива станций a, размера n

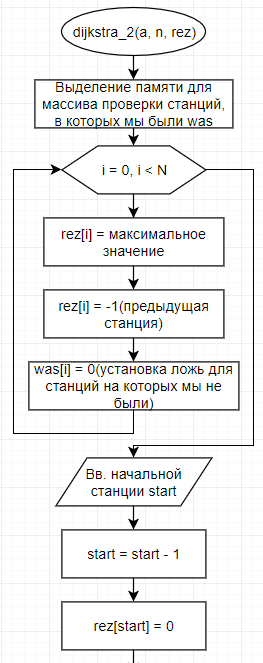


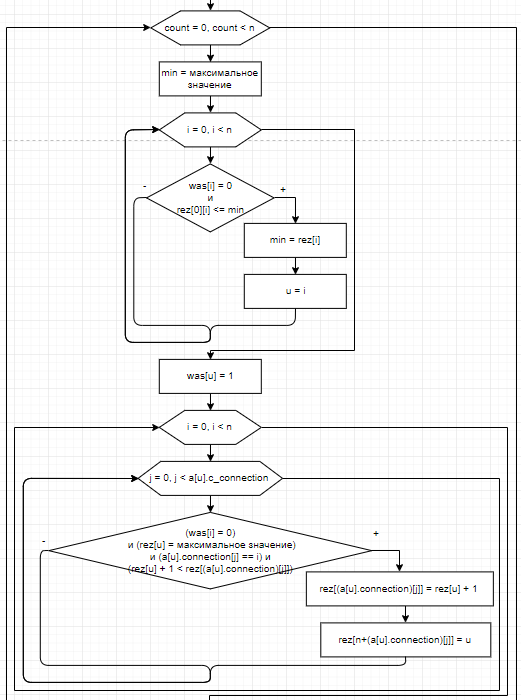


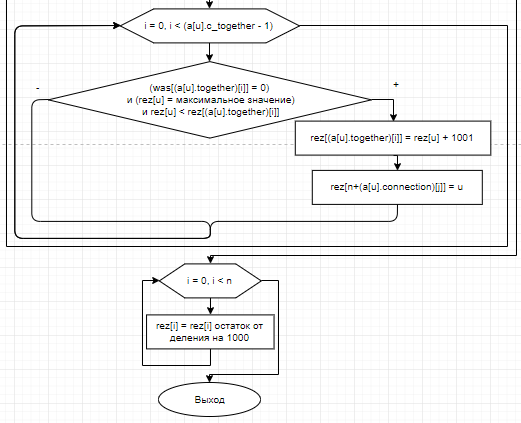


**Заголовок:** void dijkstra\_2(station \*a, int n, int \*rez)

**Назначение:** Записывает в rez путь с наименьшим числом станций по пути следования из массива станций a, размера n

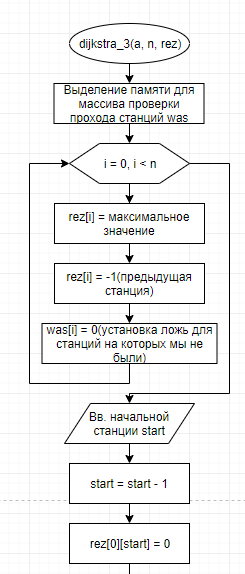


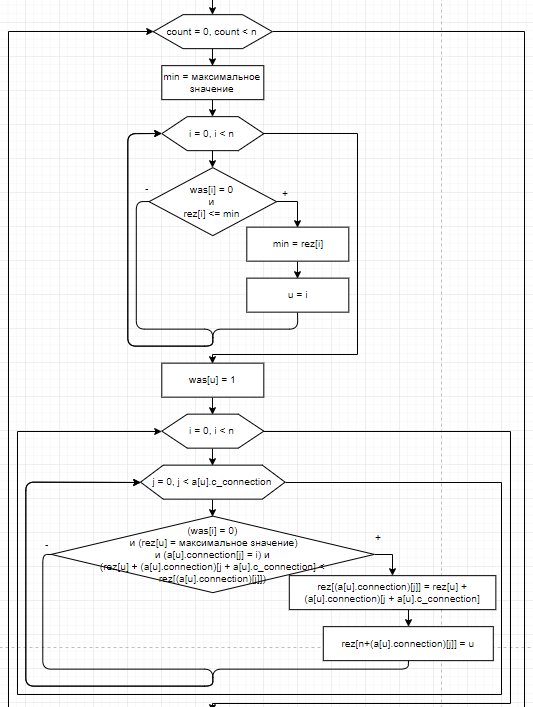


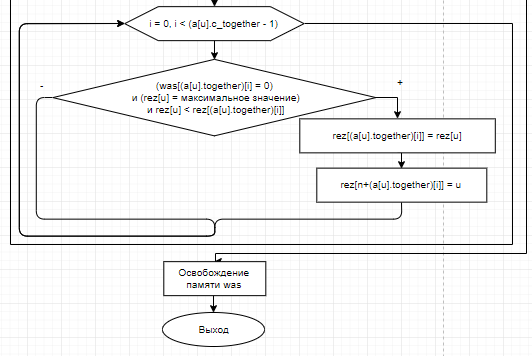


**Заголовок:** void dijkstra\_3(station \*a, int n, int \*rez)

**Назначение:** Записывает в rez путь с наименьшим временем в пути из массива станций a, размера n

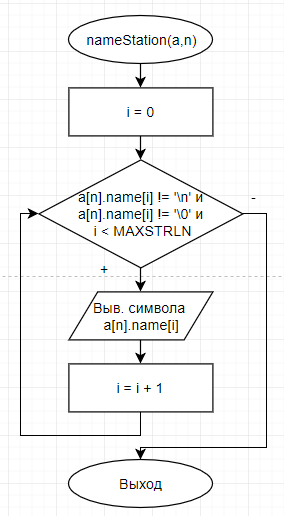






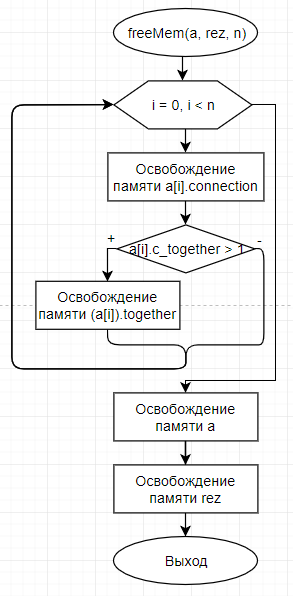
**Заголовок:** void nameStation(station \*a, int num)

**Назначение:** Выводит название станции num из массива станций a



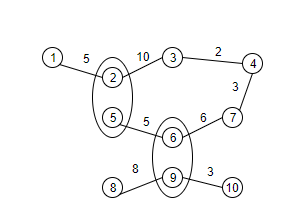
**Заголовок:** void freeMem(station \*a, int \*rez, int n)

**Назначение:** Освобождение памяти массива структур a, размера n и результата rez



# Тестовые данные

Выбор тестовых данных происходил таким образом, чтобы выполнить проверку программы. Поэтому будем использовать такую схему из 9 станций.

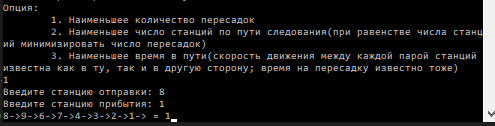


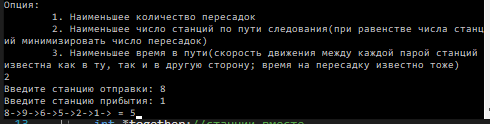
Начальной станцией возьмем 8, а конечной 1.

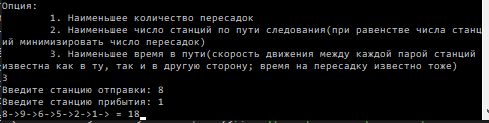
1. Наименьшее количество пересадок по пути 8->9->6->7->4->3->2->1 = 1

2. Наименьшее число станций по пути следования (при равенстве числа станций минимизировать число пересадок) 8->9->6->5->2->1 = 5

3. Наименьшее время в пути 8->9->6->5->2->1 = 18







# Модули

Модуль Graf

В модуле хранятся структура станций, а также функции ввода и обработки графа.

Заголовочный файл Graf.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS //для разрешения использования старых функций

#include <stdio.h>

#include <malloc.h>

#include <limits.h>

#define MAXSTRLN 40//максимальная длинна строки

typedef struct {//структура станции

char name[MAXSTRLN];//название станции

int \*connection; // указатель на станции связи

int c\_connection;//кол-во связей со станциями

int \*together;// указатель на общие станции

int c\_together;//кол-во общих станций

} station;

void input(station \*\*a, int n);//ввод списка смежности(данных о метро)

void inputManually(station \*a, int n);//ввод вручную

void inputManuallyName(station \*a, int n);//ввод вручную названий станций

void inputManuallyConnection(station \*a, int n);//ввод вручную связей между станциями

void inputManuallyTogether(station \*a, int n);//ввод вручную общих станций

void inputFile(station \*a, int n, char \*path1, char \*path2, char \*path3);//ввод из файла

void inputFileName(station \*a, int n, char \*path);//ввод из файла названий станций

void inputFileConnection(station \*a, int n, char \*path);//ввод из файла связей между станциями

void inputFileTogether(station \*a, int n, char \*path);//ввод из файла общих станций

void gets(char \*str);//считывает строку с консоли

void output(station \*a, int n, int \*rez); //вывод результата

void out(station \*a, int n, int \*rez, int num);//рекурсия вывод результата

void dijkstra\_1(station \*a, int n, int \*rez);//Наименьшее количество пересадок

void dijkstra\_2(station \*a, int n, int \*rez);

void dijkstra\_3(station \*a, int n, int \*rez);//наименьшее время в пути(алгоритм Дейкстры)

void nameStation(station \*a, int n);//вывод названия станции

void freeMem(station \*a, int \*rez, int n);//освобождение памяти

int gorn(char \*str);//преобразование строки в число

void reorg(char \*str);//преобразование строки в слово

void getPath(char \*path, int \*n, char \*name, char \*connection, char \*together);//получение файлов схемы метро

# Заключение

В ходе выполнения курсовой работы были получены и укреплены навыки при работе с динамической памятью, разбиении программы на файлы, в сфере разработки алгоритмов, написания и отладки программ для решения прикладных задач с использованием языка С, использования структуры данных граф, а также создания программ, состоящих из нескольких модулей, работой с потоками ввода и вывода.

# Используемая литература

1. Ю.Д. Рязанов. «Дискретная математика: учеб. пособие.» 2-е изд.Изд-во БГТУ, 2016.
2. А. Бхаргава. «Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих» изд.Питер, 2017.

# Приложение

Исходный файл Graf.cpp

#include "Graf.h"

void input(station \*\*a, int \*n) {//ввод списка смежности(данных о метро)

char c;

puts("Вручную или из файла(m/f): ");

getchar();

c = getchar();

if (c == 'm') {

printf("Введите количество станций метро: ");

scanf("%d", &n);

\*a = (station\*)malloc(sizeof(station) \* (\*n));

inputManually(\*a, \*n);//ввод вручную

}

else {//работа с файлами

FILE \*f;

char path1[MAXSTRLN];//файл названий

char path2[MAXSTRLN];//файл связей

char path3[MAXSTRLN];//файл общих станций

printf("Выберите схему метро СПБ/Тест (S/T)\n");

scanf("%c", &c);

if (c == 'S') {

char path[] = "SPB.txt";

getPath(path, n, path1, path2, path3);

}

else {

char path[] = "Test.txt";

getPath(path, n, path1, path2, path3);

}

\*a = (station\*)malloc(sizeof(station) \* (\*n));

inputFile(\*a, \*n, path1, path2, path3);//ввод из файла

}

return;

}

void getPath(char \*path, int \*n, char \*name, char \*connection, char \*together) {

FILE \*f = fopen(path, "r");

char num[MAXSTRLN];

fgets(num, MAXSTRLN, f);

\*n = gorn(num);

fgets(name, MAXSTRLN, f);

reorg(name);

fgets(connection, MAXSTRLN, f);

reorg(connection);

fgets(together, MAXSTRLN, f);

reorg(together);

fclose(f);

}

void inputManually(station \*a, int n) {//ввод вручную

inputManuallyName(a, n);//ввод вручную названий станций

inputManuallyConnection(a, n); //ввод вручную связей между станциями

inputManuallyTogether(a, n);//ввод вручную общих станций

return;

}

void inputManuallyName(station \*a, int n) {//ввод вручную названий станций

printf("Введение названий станций\n");

getchar();//символ \n

for (int i = 0; i < n; i++) {

printf("\t%d - ", i + 1);//выводим номер станции

gets(a[i].name);//считываем название станции с консоли

}//введение названия станций

return;

}

void inputManuallyConnection(station \*a, int n) {//ввод вручную связей между станциями

for (int i = 0; i < n; i++) {

printf("Cтанции связи с %d \n", i + 1);//выводим номер станции

printf("Введите количество станций связанных с ней: ");//доп информация

scanf("%d", &a[i].c\_connection);//введение количества станций

a[i].connection = (int\*)malloc(sizeof(int) \* 2 \* a[i].c\_connection);//выделение памяти для дуг(путей и времени)

for (int j = 0; j < a[i].c\_connection; j++) {

printf("%d -> ", i + 1);//вывод номера дуги

scanf("%d", &((a[i].connection)[j]));//считывание прихода дуги

(a[i].connection)[j] -= 1;

printf("Введите время прохождения между станциями: ");

scanf("%d", &((a[i].connection)[a[i].c\_connection + j]));//ввод времени прохода по дуге

}

}

return;

}

void inputManuallyTogether(station \*a, int n) {//ввод вручную общих станций

printf("Введение общих станций(переходов между линиями)\n");

for (int i = 0; i < n; i++) {

printf("Станция %d\n", i + 1);//выводим номер станции

printf("Введите количество общих станций: ");

scanf("%d", &a[i].c\_together);//вводим количество общих станций

if (a[i].c\_together > 1) {//нет смысла выделять память, если станция одиночка

a[i].together = (int\*)malloc(sizeof(int)\*a[i].c\_together);//выделение памяти для общих станций

for (int j = 0; j < a[i].c\_together - 1; j++) {

printf("Введите общие: ");

scanf("%d", &((a[i].together)[j]));//ввод общих станций

(a[i].together)[j] -= 1;

}

}

}

return;

}

void inputFile(station \*a, int n, char \*path1, char \*path2, char\* path3) {//ввод из файла

inputFileName(a, n, path1);//ввод из файла названий станций

inputFileConnection(a, n, path2);//ввод из файла связей между станциями

inputFileTogether(a, n, path3);//ввод из файла общих станций

return;

}

void inputFileName(station \*a, int n, char \*path) {//ввод из файла названий станций

FILE \*f;//переменная для файла

//char path[MAXSTRLN] = "nameMetro.txt";//----------------------------------------------------

f = fopen(path, "r");//Подготовка f, для чтения файла path(текстовый файл)

printf("Названия станций: \n");

for (int i = 0; i < n; i++) {

fgets(a[i].name, MAXSTRLN, f);//считывание названий станций

printf("%d - ", i + 1);

puts(a[i].name);

}

fclose(f);

return;

}

void inputFileConnection(station \*a, int n, char \*path) {//ввод из файла связей между станциями

FILE \*f;

//char path[MAXSTRLN] = "SpbNetwork.bin";//---------------------------------------------------

f = fopen(path, "rb");//Подготовка f, для чтения файла path(бинарный файл)

for (int i = 0; i < n; i++) {

fread(&(a[i].c\_connection), sizeof(int), 1, f);//количество станций

printf("Введите количество станций связанных с ней: %d", a[i].c\_connection);

printf("\n");

a[i].connection = (int\*)malloc(sizeof(int) \* 2 \* a[i].c\_connection);//выделение памяти для дуг(путей и времени)

for (int j = 0; j < a[i].c\_connection; j++) {

fread(&(a[i].connection)[j], sizeof(int), 1, f);//связь со станцией

printf("%d -> %d", i + 1, (a[i].connection)[j] + 1);

printf("\n");

fread(&((a[i].connection)[a[i].c\_connection + j]), sizeof(int), 1, f);//время до станции

printf("Введите время прохождения между станциями: %d", (a[i].connection)[a[i].c\_connection + j]);

printf("\n");

}

}

fclose(f);

return;

}

void inputFileTogether(station \*a, int n, char \*path) {//ввод из файла общих станций

FILE \*f;

//char path[MAXSTRLN] = "general.bin";//-----------------------------------------------------

f = fopen(path, "rb");//Подготовка f, для чтения файла path(бинарный файл)

for (int i = 0; i < n; i++) {

fread(&a[i].c\_together, sizeof(int), 1, f);//считываем количество общих станций

if (a[i].c\_together > 1) {

a[i].together = (int\*)malloc(sizeof(int)\*a[i].c\_together);

for (int j = 0; j < a[i].c\_together - 1; j++) {

fread(&((a[i].together)[j]), sizeof(int), 1, f);//считываем с файла в структуру информацию о общих станциях

(a[i].together)[j] -= 1;

}

}

}

fclose(f);

return;

}

void gets(char \*str) {

char c;

int i = 0;

while ((c = getchar()) != '\n')

str[i++] = c;

str[i] = '\0';

}//считываем строку

void dijkstra\_1(station \*a, int n, int \*rez) { //поиск самого короткого пути по времени

int start;//станция отправки

int u;//самый минимальный не проверенный

int min;//сохранение минимального значения

int i, j, count;//счётчики

int \*was = (int\*)malloc(sizeof(int)\*n);//пройденные станции

for (i = 0; i < n; i++) {//создание пустых массивов данных

rez[i] = INT\_MAX;//установка максимального значения для времени пути

rez[n + i] = -1;//пустое значение для предыдущей станции

was[i] = 0;//установка ложного значения

}

printf("Введите станцию отправки: ");

scanf("%d", &start);//введение станции отпавки

start -= 1;//установка на массив

rez[start] = 0;//начальная станция

for (count = 0; count < n; count++) {

min = \_CRT\_INT\_MAX;//максимальное значение инт

for (i = 0; i < n; i++)

if (!was[i] && (rez[i] <= min)) {

min = rez[i];

u = i;//Ищем значение старта

}

was[u] = 1;//были на станции u

for (i = 0; i < n; i++)

for (j = 0; j < a[u].c\_connection; j++) {

if (!was[i] && rez[u] != INT\_MAX && a[u].connection[j] == i && rez[u] <= rez[(a[u].connection)[j]]) {//самый бррр, два цикла, из-за сдвига динамического и статического массива

rez[(a[u].connection)[j]] = rez[u];//считаем пересадки

rez[n + (a[u].connection)[j]] = u;//запоминаем предыдущую станцию

}

}

for (i = 0; i < (a[u].c\_together - 1); i++) {

if (!was[(a[u].together)[i]] && rez[u] != INT\_MAX && rez[u] < rez[(a[u].together)[i]]) {

rez[(a[u].together)[i]] = rez[u] + 1;//считаем пересадки

rez[n + (a[u].together)[i]] = u;//запоминаем предыдущую станцию

}

}

}

free(was);

return;

}

void dijkstra\_2(station \*a, int n, int \*rez) { //поиск самого короткого пути по времени

int start;//станция отправки

int u;//самый минимальный не проверенный

int min;//сохранение минимального значения

int i, j, count;//счётчики

int \*was = (int\*)malloc(sizeof(int)\*n);//пройденные станции

for (i = 0; i < n; i++) {//создание пустых массивов данных

rez[i] = INT\_MAX;//установка максимального значения

rez[n + i] = -1;//пустое значение для предыдущей станции

was[i] = 0;//установка ложного значения

}

printf("Введите станцию отправки: ");

scanf("%d", &start);//введение станции отпавки

start -= 1;//установка на массив

rez[start] = 0;

for (count = 0; count < n; count++) {

min = \_CRT\_INT\_MAX;//максимальное значение инт

for (i = 0; i < n; i++)

if (!was[i] && (rez[i] <= min)) {

min = rez[i];

u = i;//Ищем значение старта

}

was[u] = 1;

for (i = 0; i < n; i++)

for (j = 0; j < a[u].c\_connection; j++) {

if (!was[i] && rez[u] != INT\_MAX && a[u].connection[j] == i && (rez[u] + 1)%1000 < (rez[(a[u].connection)[j]])%1000) {//самый бррр, два цикла, из-за сдвига динамического и статического массива

rez[(a[u].connection)[j]] = rez[u] + 1;

rez[n + (a[u].connection)[j]] = u;

}

}

for (i = 0; i < (a[u].c\_together - 1); i++) {

if (!was[(a[u].together)[i]] && rez[u] != INT\_MAX && rez[u]%1000 < rez[(a[u].together)[i]]%1000) {

rez[(a[u].together)[i]] = rez[u] + 1001;

rez[n + (a[u].together)[i]] = u;

}

}

}

for (i = 0; i < n; i++)

rez[i] = rez[i] % 1000;

free(was);

return;

}

void dijkstra\_3(station \*a, int n, int \*rez) { //поиск самого короткого пути по времени

int start;//станция отправки

int u;//самый минимальный не проверенный

int min;//сохранение минимального значения

int i, j, count;//счётчики

int \*was = (int\*)malloc(sizeof(int)\*n);//пройденные станции

for (i = 0; i < n; i++) {//создание пустых массивов данных

rez[i] = INT\_MAX;//установка максимального значения

rez[n + i] = -1;//пустое значение для предыдущей станции

was[i] = 0;//установка ложного значения

}

printf("Введите станцию отправки: ");

scanf("%d", &start);//введение станции отпавки

start -= 1;//установка на массив

rez[start] = 0;

for (count = 0; count < n; count++) {

min = \_CRT\_INT\_MAX;//максимальное значение инт

for (i = 0; i < n; i++)

if (!was[i] && (rez[i] <= min)) {

min = rez[i];

u = i;//Ищем значение старта

}

was[u] = 1;

for (i = 0; i < n; i++)

for (j = 0; j < a[u].c\_connection; j++) {

if (!was[i] && rez[u] != INT\_MAX && a[u].connection[j] == i && rez[u] + (a[u].connection)[j + a[u].c\_connection] < rez[(a[u].connection)[j]]) {//самый бррр, два цикла, из-за сдвига динамического и статического массива

rez[(a[u].connection)[j]] = rez[u] + (a[u].connection)[j + a[u].c\_connection];

rez[n + (a[u].connection)[j]] = u;

}

}

for (i = 0; i < (a[u].c\_together - 1); i++) {

if (!was[(a[u].together)[i]] && rez[u] != INT\_MAX && rez[u] < rez[(a[u].together)[i]]) {

rez[(a[u].together)[i]] = rez[u];

rez[n + (a[u].together)[i]] = u;

}

}

}

free(was);

return;

}

void output(station \*a, int n, int \*rez) {

int end;

printf("Введите станцию прибытия: ");

scanf("%d", &end);

end -= 1;

out(a, n, rez, end);

printf(" = %d", rez[end]);

return;

}

void out(station \*a, int n, int \*rez, int num) {

if (num == -1)

return;

else {

out(a, n, rez, rez[n + num]);

nameStation(a, num);

printf("->");

}

}

void nameStation(station \*a, int n) {

int i = 0;

while (a[n].name[i] != '\n' && a[n].name[i] != '\0' && i < MAXSTRLN) {

printf("%c", a[n].name[i]);

i++;

}

return;

}

void freeMem(station \*a, int \*rez, int n) {

for (int i = 0; i < n; i++) {

free((a[i]).connection);

if (a[i].c\_together > 1)

free((a[i]).together);

}

free(a);

free(rez);

}

int gorn(char \*str) {

int S = str[0] - '0';

for (int i = 1; str[i] != '\0' && str[i] != '\n' && i < MAXSTRLN; i++)

S = S \* 10 + str[i] - '0';

return S;

}

void reorg(char \*str) {

int i;

for (i = 0; str[i] != '\n' && str[i] != '\0' && i < MAXSTRLN; i++)

;

str[i] = '\0';

}

Главная программа

#define \_CRT\_SECURE\_NO\_WARNINGS //для разрешения использования старых функций

#include <stdio.h>

#include <locale.h>

#include <malloc.h>

#include "Graf.h"

int main() {//главная функция

setlocale(LC\_ALL, "Rus");

int n;

station \*a;

input(&a, &n);

printf("Опция: \n\t1. Наименьшее количество пересадок\n\t2. Наименьшее число станций по пути следования(при равенстве числа станций минимизировать число пересадок)\n\t3. Наименьшее время в пути(скорость движения между каждой парой станций известна как в ту, так и в другую сторону; время на пересадку известно тоже)\n");

int t;

scanf("%d", &t);

int \*rez = (int\*)malloc(sizeof(int) \* 2 \* n);

switch (t) {

case(1):

dijkstra\_1(a, n, rez);

break;

case(2):

dijkstra\_2(a, n, rez);

break;

case(3):

dijkstra\_3(a, n, rez);

}

output(a, n, rez);

freeMem(a, rez, n);

getchar();

getchar();

return 0;

}